Lab 12: Blackjack

Joshua Pollock

Lab Section 1

December 11, 2016

**Problem Statement:**

The problem that was given to us was to create two classes that implemented code from Lab 10, Casino Night. These two classes would be a way to play blackjack. There were afew rules to blackjack that we were to follow. These rules were:

* Face cards are worth 10 points. Aces are worth either 1 or 11 points, beneficial to the owner of the hand. Other cards are worth their rank value
* The player is dealt 2 face up cards
* The dealer is dealt a face-down card and a face-up card
* If both player blackjack (get 21 on the deal), the game ends in a tie.
* If either the player or dealer have 21 and the other does not, the other wins the hand
* The player may choose to hit as many times as they want, or stand.
* If a player hits, a new card is drawn. If the value exceeds 21, the player busts/loses.
* Once the player stands, the dealer will reveal their face down card. And will choose to draw if their hand is currently under 16.
* IF at any point someone’s hand goes over 21, they will bust.
* Once both players are done drawing, The highest hand wins.
* If the player wins, their wager is doubled. If they lose, they get nothing. If they push/tie, their wager is returned.

These rules will act as our constraints for this lab. We are required to use both the Blackjack and BlackjackHand class in our final code. There is possible extra credit by creating a simulation for the Blackjack class.

**Planning:**

The first thing we did was create a UML diagram for our two classes. These would end up being our framework for the code.

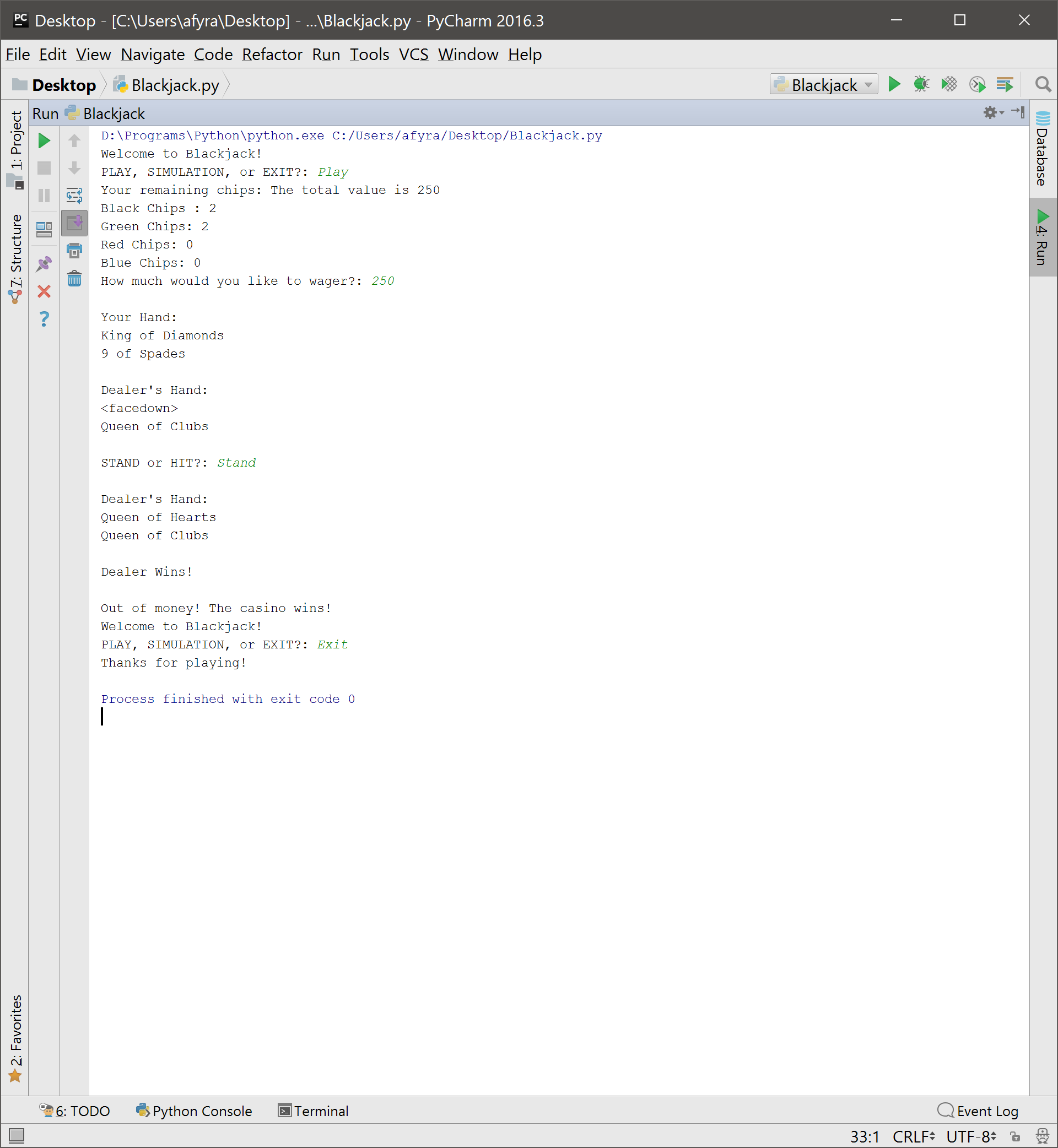
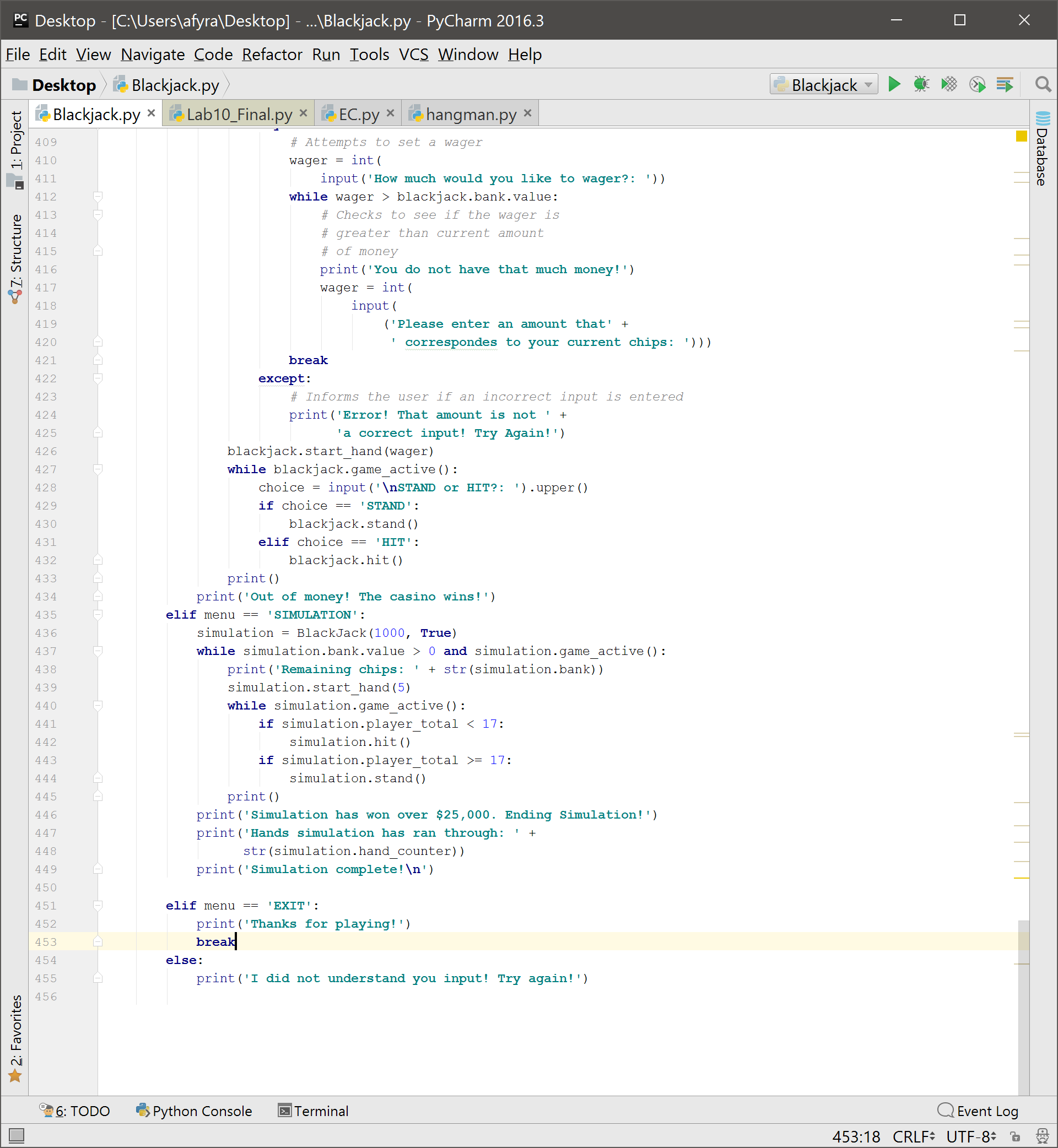
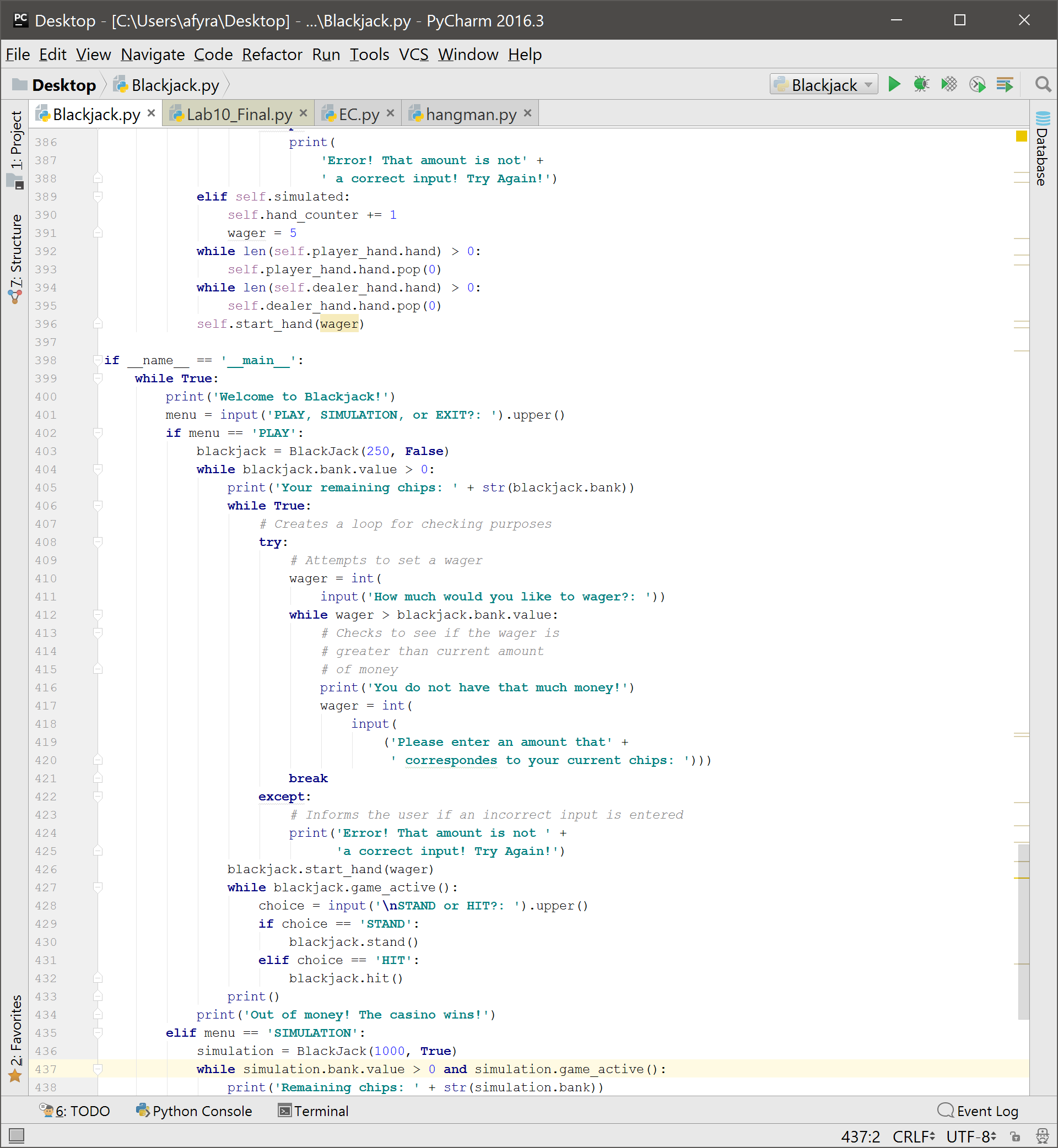
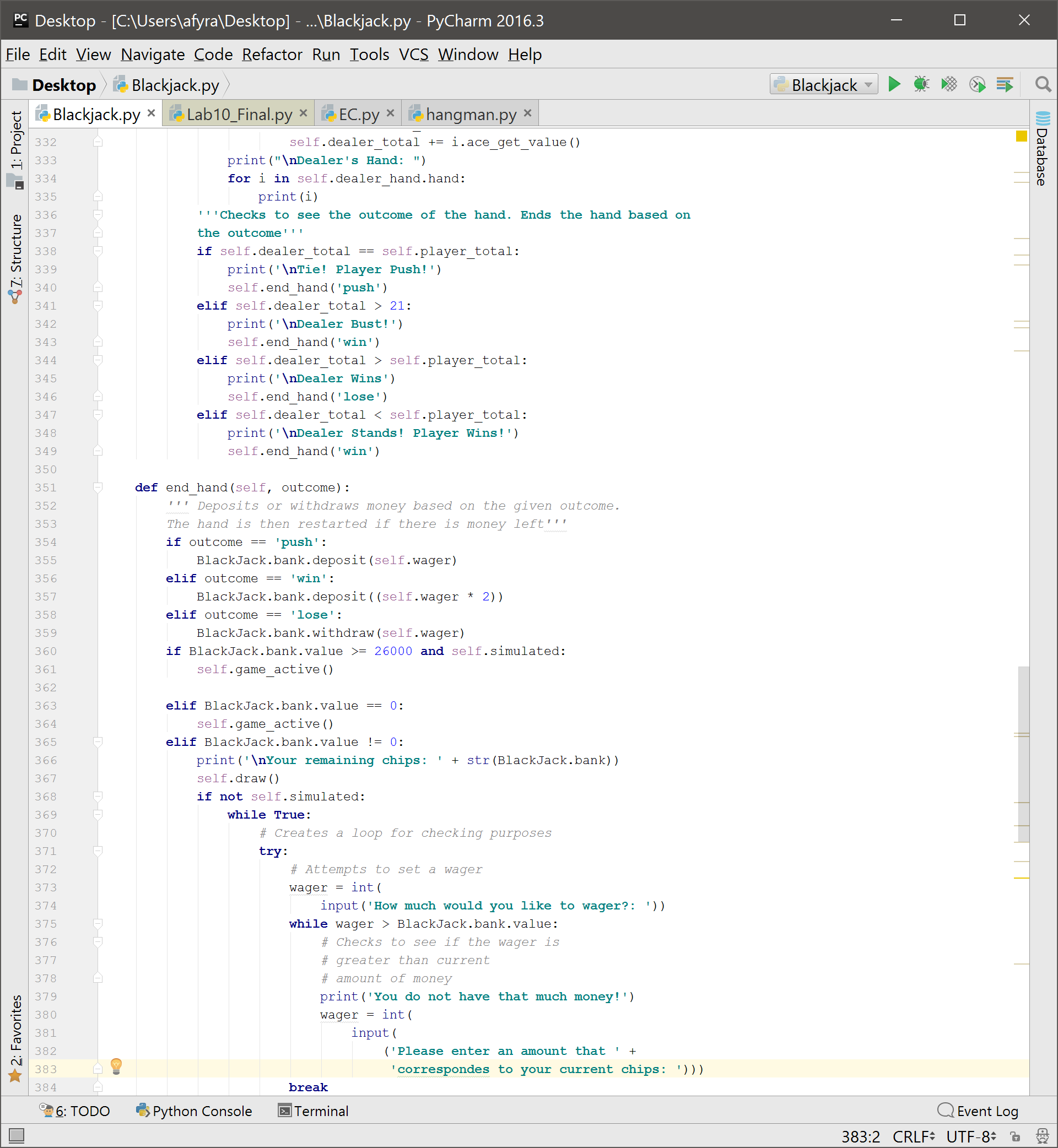
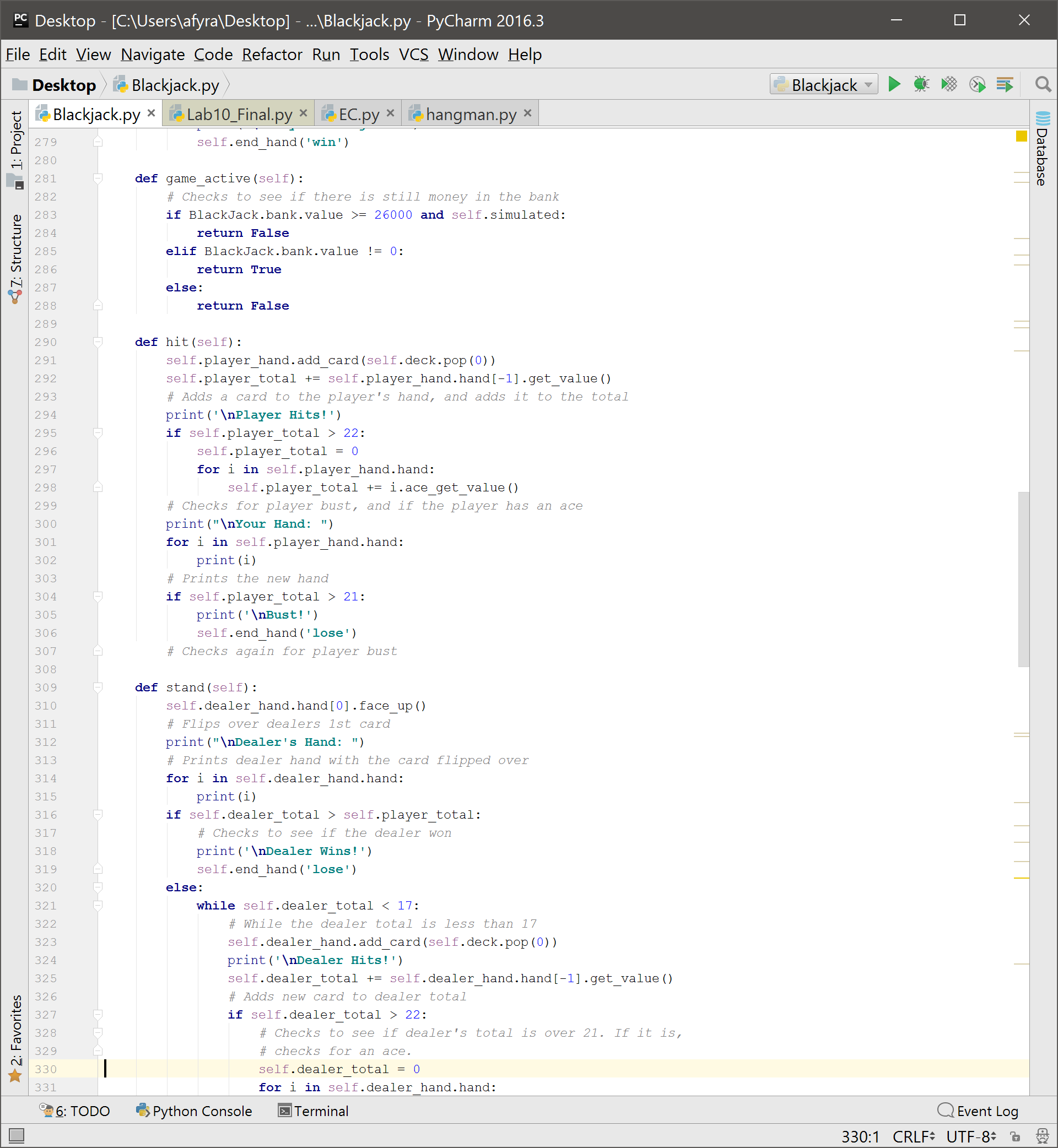
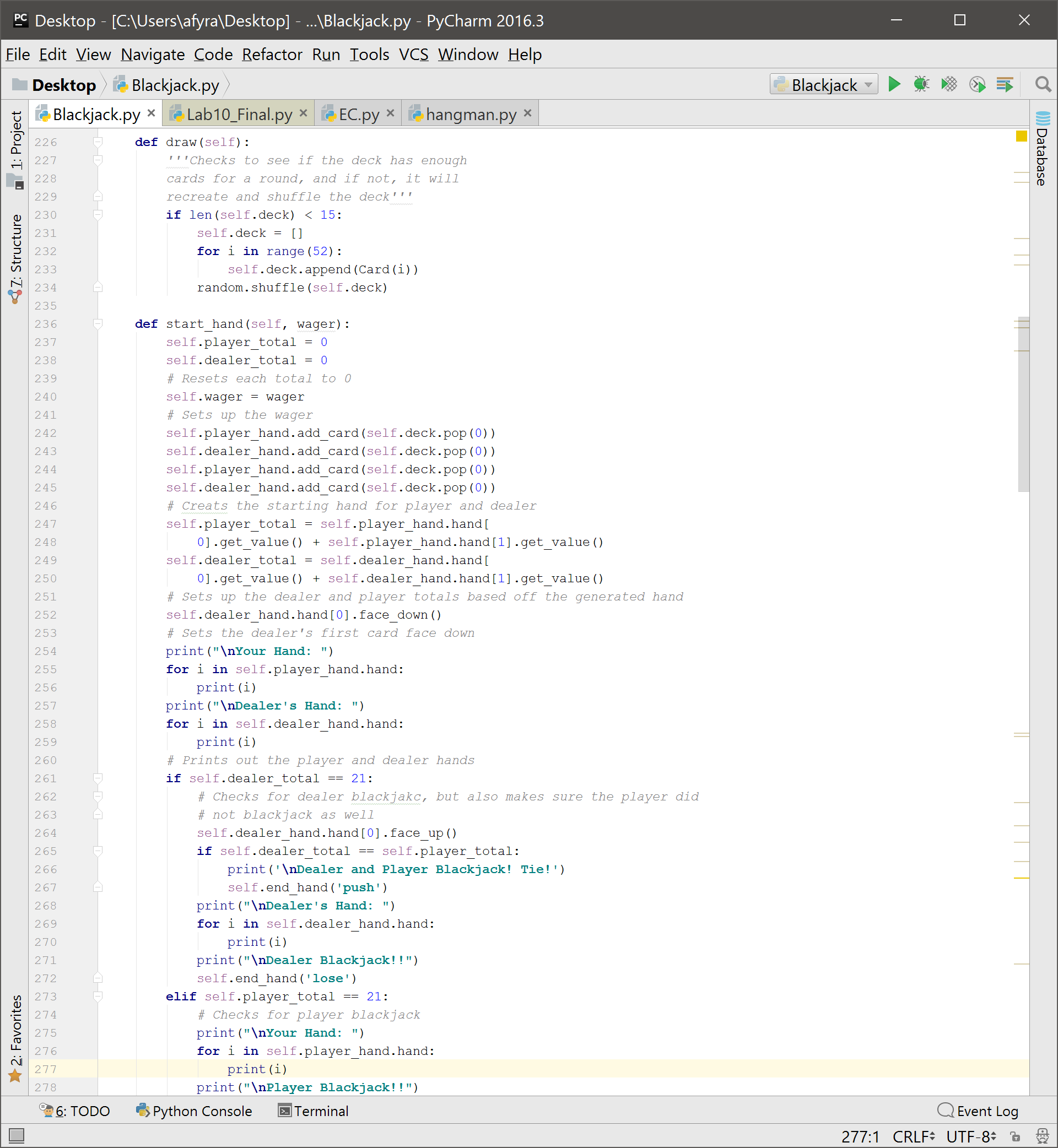
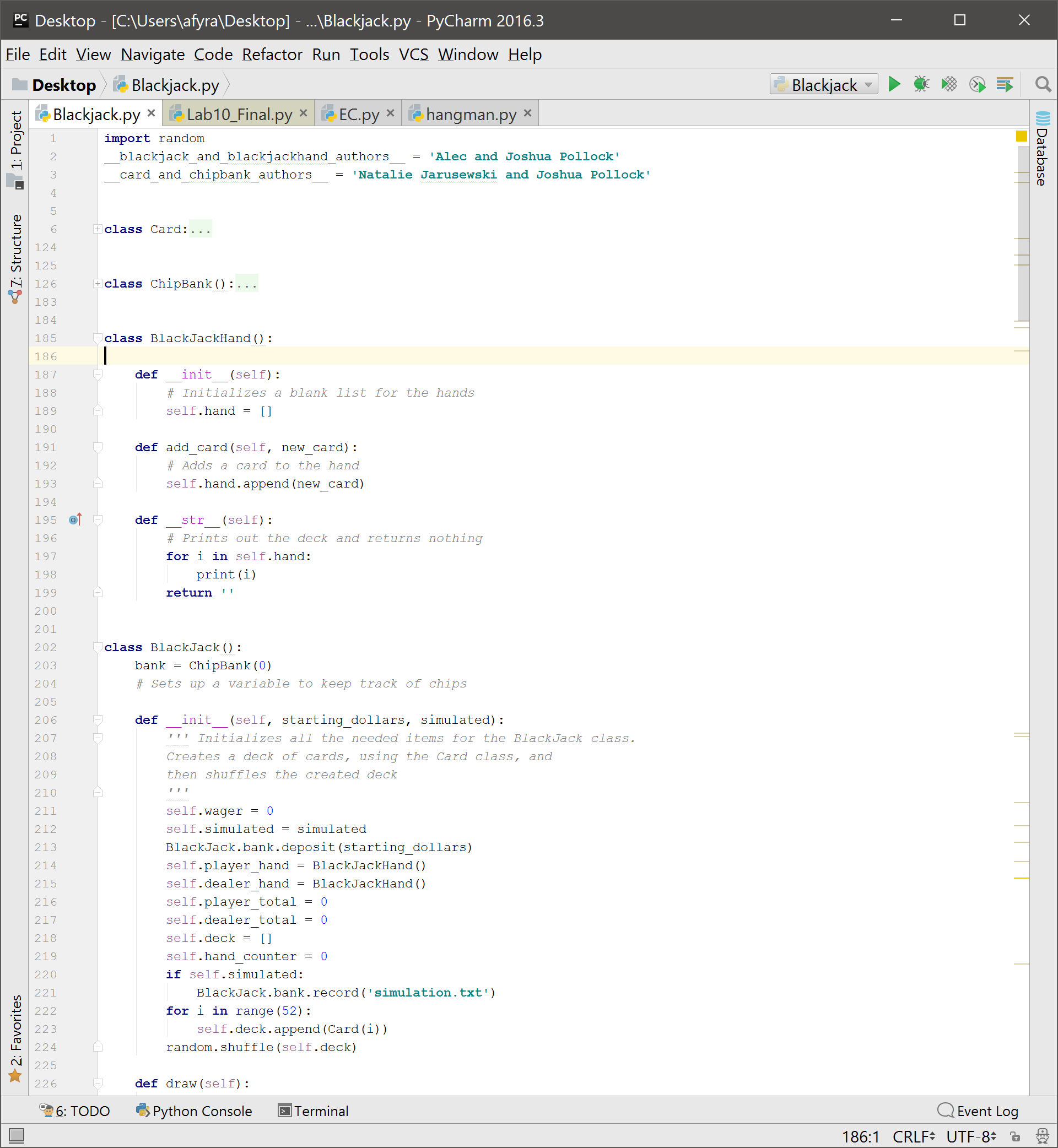
|  |
| --- |
| Blackjack() |
| \_\_init(starting\_dollars, simulated)  Draw()  Start\_hand(wager)  Game\_active()  Hit()  Stand()  End\_hand(outcome) |
| Bank  Self.player\_hand  Self.dealer\_hand  Self.simulated  Self.player\_total  Self.dealer\_total  Self.deck  Self.hand\_counter |

|  |
| --- |
| BlackjackHand() |
| \_\_init\_\_()  Add\_card(new\_card)  \_\_str\_\_() |
| Self.hand |

Some assumptions to be made is that the user’s input may not always be correct. We will try to put in preventative measures to keep the code from erroring. This may mean we have to change the test code a little. This will most likely be accomplished by a few if statements and try/except blocks. We plan to use many different types of loops and operations. For example, we will be using plenty of while, for, and if statements throughout our code.

**Implementation and Testing:**

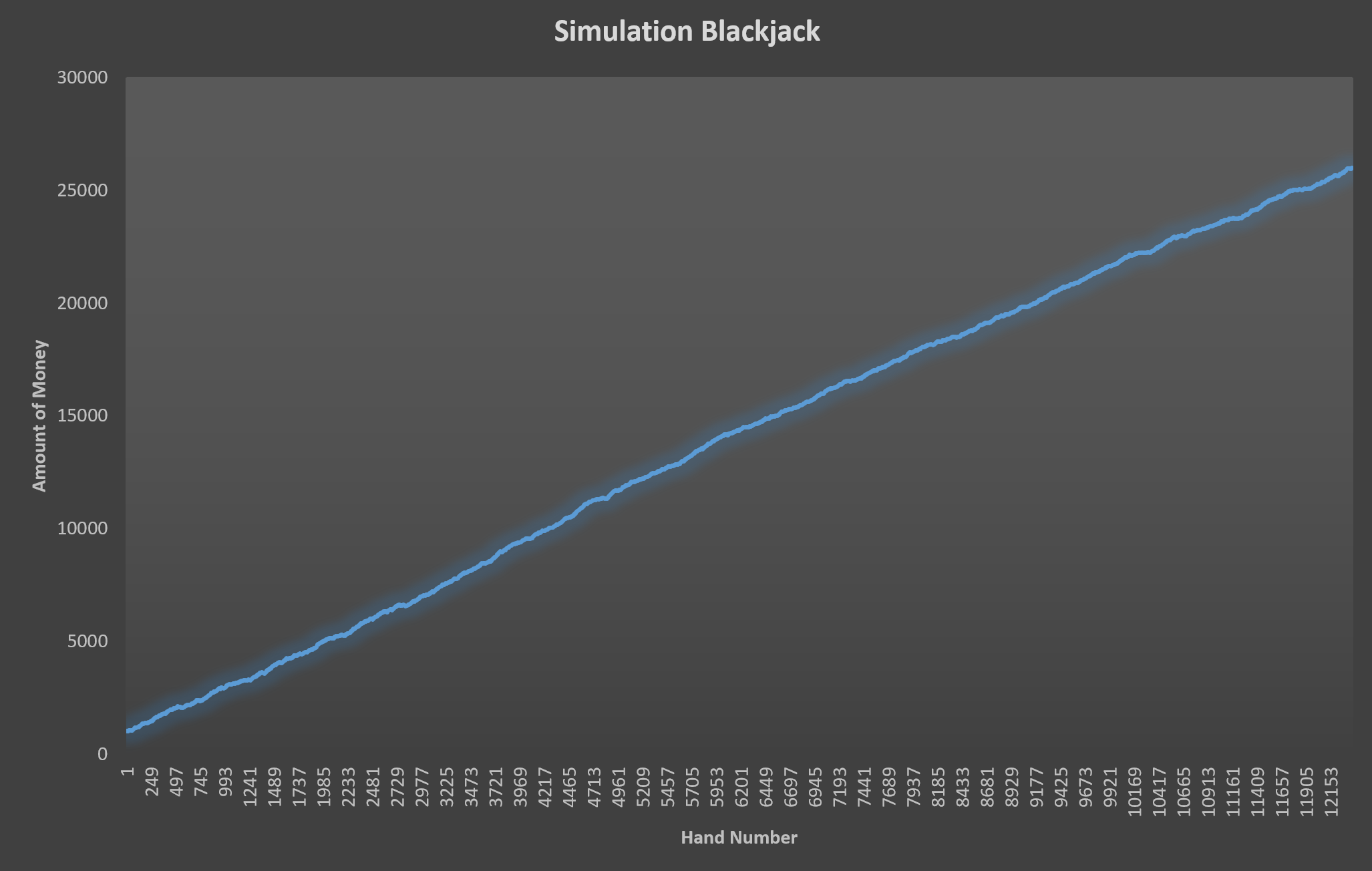
Implementing the code was quite simple. The skeleton of the classes was given to us, and all we needed to do was put in the code to return the desired item. We found that using pop on lists was very efficient for our code. At first, we did not implement the use of the BlackjackHand class. To fulfill the requirement of using this class, we decided to use it instead of setting self.player\_hand and self.dealer\_hand to an empty list.

**Reflection/Refactoring:**

Overall I am very pleased with how the code turned out. Some changes that could be made is the layout of the print statements. Currently it uses up quite a bit of screen space, but could easily be condensed down to a couple of lines instead. This would only be an aesthetic change and would not change the function of the code. To possibly condense the code, not only could we remove some lines of code, but we could also set up a way to import the Card and Chipbank class. This would reduce the amount of lines by about 200+ lines. The current solution currently functions as desired and I am quite satisfied with it.

**Extra Credit:**

There was possible extra credit for creating a functioning simulation for the code. I was able to Create a functioning simulation. I had to put a limit on how much money it could make, otherwise it would be like an infinite loop. I set the simulation to quit after it made $25,000. It then prompts the user that the simulation has made over $25,000 and tells the user how many hands it completed. If I wanted to clean up the simulation, I could make it so it does not print out what is going on. Currently it floods the screen full of print statements. Below is the graph of the money per round.



The code also creates a text file called ‘simulation.txt’ for the user.

